A new approach to sequence comparison : Normalized sequence alignment

Abdullah N. Arslan* and Ömer Eğecioğlu†
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
{*arslan, omer*}@*cs.ucsb.edu*

Pavel A. Pevzner
Department of Computer Science and Engineering
University of California, San Diego, San Diego, CA 92093
ppevzner@cs.ucsd.edu

Abstract

The Smith-Waterman algorithm for local sequence alignment is one of the most important techniques in computational molecular biology. This ingenious dynamic programming approach was designed to reveal the highly conserved fragments by discarding poorly conserved initial and terminal segments. However, the existing notion of local similarity has a serious flaw: it does not discard poorly conserved intermediate segments. The Smith-Waterman algorithm finds the local alignment with maximal *score* but it is unable to find local alignment with maximum *degree* of similarity (e.g., maximal percent of matches). Moreover, there is still no efficient algorithm that answers the following natural question: do two sequences share a (sufficiently long) fragment with more than 70% of similarity? As a result, the local alignment sometimes produces a mosaic of well-conserved fragments artificially connected by poorly-conserved or even unrelated fragments. This may lead to problems in comparison of long genomic sequences and comparative gene prediction as recently pointed out by Zhang *et al.* (1999). In this paper we propose a new sequence comparison algorithm (*normalized local alignment*) that reports the regions with maximum degree of similarity. The algorithm is based on fractional programming and its running time is $O(n^2 \log n)$. In practice, normalized local alignment is only 3-5 times slower than the standard Smith-Waterman algorithm.

Keywords: Sequence alignment, normalized local sequence alignment, algorithm, dynamic programming, fractional programming, ratio maximization.

1 Introduction

Gene prediction in human genome often amounts to using related proteins from other species as clues for finding exon-intron structures (Gelfand *et al.*, 1996; Pachter *et al.*, 1999; Birney *et al.*,

*Supported in part by a UCSB–COR grant.
†Supported in part by NSF Grant No. CCR–9821038.

1996). Recently, a related paradigm, motivated by availability of complete genomes, has emerged (Batzoglou *et al.*, 2000; Bafna and Huson, 2000; Novichkov *et al.*, 2000). In this new approach, human genes are predicted based on other (e.g., mouse) un-annotated genomic sequences. The idea of this method is that similarity between nucleotide sequences of related human and mouse exons is 85% on average, while similarity between introns is 35% on average. This observation motivates the following simple approach: use local alignment algorithm (Smith and Waterman, 1981) to find the most similar segments in human and mouse genomic sequences and use these fragments as potential exons at the further stages.

Unfortunately, this approach faces serious difficulties. Smith-Waterman algorithm was developed 20 years ago for a different problem and it is not well suitable for sequence comparison at genomic scale. Surprisingly enough, we still don't have an efficient algorithm that finds the local alignment with the best degree of sequence similarity. The following example illustrates this point.

It is well-known that the statistical significance of the local alignment depends on both its score and length (Altschul and Erickson, 1986; Altschul and Erickson, 1988). However, the score of a local alignment is not normalized over the length of the matching region. As a result, a local alignment with score 1,000 and length 10,000 (*long alignment*) will be chosen over a local alignment with score 998 and length 1,000 (*short alignment*), although the latter one is probably more important biologically. Moreover, if the corresponding alignment paths overlap, the more biologically important "short" alignment will not be detected even by suboptimal sequence alignment algorithm (*shadow effect*). Another unfortunate property of the Smith-Waterman algorithm is that it was designed to exclude non-similar initial and terminal fragments in sequence alignment but it was not designed to exclude non-similar internal fragments. This flaw with Smith-Waterman local similarity approach (Figure 1) leads to inclusion of arbitrarily poor internal fragments (*mosaic effect*). As a result, applications of the Smith-Waterman algorithm to comparison of related genomes (particularly with short introns as *C. elegans* and *C. briggsae*) may lead to problems (Zhang *et al.*, 1999).

Figure 1: The inclusion of an arbitrarily poor region in an alignment (Zhang *et al.*, 1999). If a region of negative score $-X$ is sandwiched between two regions scoring more than X, then the Smith-Waterman algorithm will join the three regions into a single alignment that may not be biologically adequate.

The attempts to fix the problem of mosaic effect undertaken by Goad and Kanehisa (1982) (who introduced alignment with minimal mismatch density) and Sellers (1984) did not lead to successful algorithms and were later abandoned. The mosaic effect was first analyzed by Webb Miller (personal communication) and led to some studies trying to fix this problem at the post-processing stage (Huang *et al.*, 1994; Zhang *et al.*, 1999). Zhang *et al.* (1999) proposed to decompose a local alignment into sub-alignments that avoid the mosaic effect. However, the post-processing approach may miss the alignments with the best degree of similarity if the Smith-Waterman algorithm missed them. As a result, highly similar fragments may be ignored if they are not parts of larger alignments dominating other local similarities. Another approach to fixing the problems with the Smith-Waterman algorithm is based on the notion of X-*drop*, a region within an alignment that scores below X. The alignments that contain no X-drops are called X-*alignments*. Although X-alignments are expensive to compute in practice, Altschul *et al.* (1997) and Zhang *et al.* (1998) used some heuristics for searching databases with this approach. Other attempts to fix the problem

of mosaic effect involve modifications of the local alignment algorithm that allow insertions of very long gaps.

Another deficiency of the local alignment was recently revealed by Alexandrov and Solovyev (1998). They asked if the Smith-Waterman algorithm correctly finds the most biologically adequate relative in a benchmark sample of different protein families. The answer to this question was negative, and Alexandrov and Solovyev (1998) "blamed" it on the fact that the Smith-Waterman algorithm does not take into account the length of the alignment. They proposed to normalize the alignment score by its length and demonstrated that this new approach leads to better protein classification. However, computing normalized scores in alignments may be very expensive when there is a constraint on length. An algorithm for a similar problem (*normalized edit distance*) uses dynamic programming to compute the minimum edit distances for all lengths (Marzal and Vidal, 1993), but requires cubic time and quadratic space. Various parallel algorithms for this problem were developed by Eğecioğlu and Ibel (1996). We want to emphasize the difference between the normalized local alignment and the previously studied normalized edit distance problem. The algorithms by Oommen and Zhang (1996), Vidal *et al.* (1995), Arslan and Eğecioğlu (1999), Arslan and Eğecioğlu (2000) do not aim to satisfy a constraint on the length, therefore they cannot directly be adapted to the the computation of normalized scores when lengths are restricted.

In this paper, we propose a new practical algorithm that produces local alignment with maximum degree of similarity by extending the ideas presented by Arslan and Eğecioğlu (1999), Arslan and Eğecioğlu (2000). To reflect the length of the local alignment in scoring, the score $s(I, J)$ of local alignment involving substrings I and J may be adjusted by dividing $s(I, J)$ by the total length of the aligned regions: $s(I, J)/(|I| + |J|)$. The *normalized local alignment problem* is to find substrings I and J that maximize $s(I, J)/(|I|+|J|)$ among all substrings I and J with $|I|+|J| \geq T$, where T is a threshold for the minimal overall length of I and J. For the same problem with no restriction on overall length, we can develop fast algorithms using *fractional programming*, however the answer to the problem would be short substrings that are not biologically meaningful. We use a slightly different objective to normalized alignment. We aim to maximize $s(I, J)/(|I| + |J| + L)$ for a given parameter L . Our purpose is to provide a way of control over the degree of normalization by varying L, and at the same time still being able to use fractional programming technique for fast computation.

The outline of this paper is as follows. We first give a formal definition of our approach to the normalized local alignment problem. We include brief information on Dinkelbach's and Megiddo's methods as we use them in our algorithms. Description of our algorithms are followed by discussion of some implementation issues and test results, and concluding remarks at the end.

2 Normalized Local Alignment

First we formulate the alignment problems we study in this paper as optimization problems involving quotients of linear functions. We are then able to use applicable optimization methods such as fractional programming to develop our algorithms for normalized local alignment.

Let $a = a_1 a_2 \cdots a_n$ and $b = b_1 b_2 \cdots b_m$ be two sequences of symbols over an alphabet Σ with $n \geq m$. The *alignment graph* $G_{a,b}$ (*edit graph* in the context of string editing) is used to represent all possible *alignments* (Waterman, 1995) between a and b . It is a directed acyclic graph having $(n + 1)(m + 1)$ lattice points (u, v) for $0 \leq u \leq n$, and $0 \leq v \leq m$ as vertices (Figure 2). The arcs of $G_{a,b}$ are divided into four types :

(1) *Horizontal arcs*: $\{((u, v - 1), (u, v)) \mid 0 \leq u \leq n,\ 0 < v \leq m\}$.

(2) *Vertical arcs*: $\{((u-1,v),(u,v)) \mid 0 < u \leq n,\ 0 \leq v \leq m\}$.

(3) *Matching diagonal arcs*: $\{((u-1,v-1),(u,v)) \mid a_u = b_v,\ 0 < u \leq n,\ 0 < v \leq m\}$

(4) *Mismatching diagonal arcs*: $\{((u-1,v-1),(u,v)) \mid a_u \neq b_v,\ 0 < u \leq n,\ 0 < v \leq m\}$

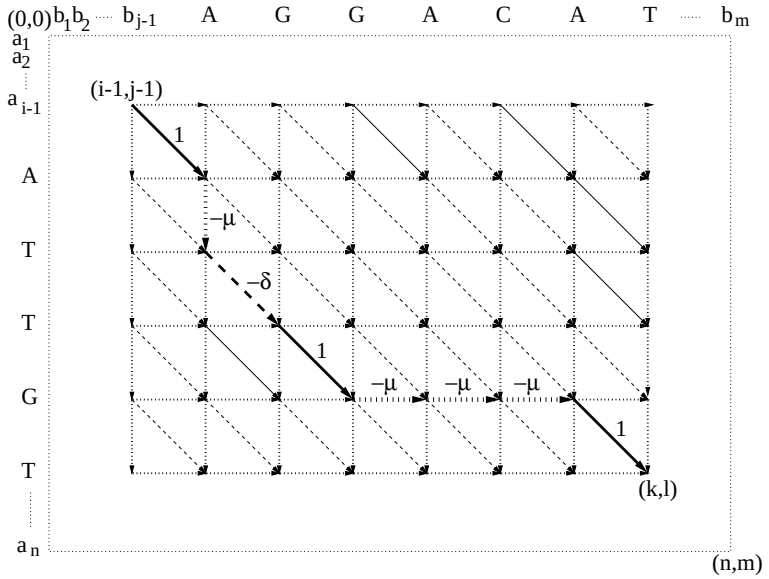

Figure 2: The alignment graph $G_{a,b}$ where $a_i \cdots a_k = ATTGT$ and $b_j \cdots b_l = AGGACAT$. Matching diagonal arcs are drawn as solid lines while mismatching diagonal arcs are shown by dashed lines. Dotted lines used for horizontal and vertical arcs correspond to indels. An example alignment path is shown. Only the weights of the arcs in this path are included.

Consider a directed path p between two vertices $(i-1, j-1)$ and (k, l) on $G_{a,b}$ where $i \leq k$ and $j \leq l$. We call each such path an *alignment path* since if we trace the arcs of p, and perform the corresponding edit operations in segment $a_i \cdots a_k$, we obtain the segment $b_j \cdots b_l$ as follows :

(1) For a horizontal arc $((u, v-1),(u, v))$, insert b_v immediately before a_u .

(2) For a vertical arc $((u-1, v),(u, v))$, delete a_u .

(3) For a mismatching diagonal arc $((u-1, v-1),(u, v))$, substitute b_v for a_u .

In the context of sequence alignment, insertions (horizontal arcs) and deletions (vertical arcs) are both called as *indel*s, and the names *match*, and *mismatch*, are used to refer to matching diagonal, and mismatching diagonal arcs.

The objective of sequence alignment is to quantify the similarity between two strings. There are various scoring schemes for this purpose. In one simple such method, the arcs of $G_{a,b}$ have weights determined by positive reals δ (*mismatch penalty*) and μ (*indel* or *gap penalty*) as shown in Figure 2. We assume that a match has a score of 1, a mismatch penalty is δ, and an indel has a penalty of μ. Existence of an alignment path with a large total weight between the vertices $(i-1, j-1)$ and (k, l) indicates a high similarity between the segments $a_i \cdots a_k$ and $b_j \cdots b_l$.

For clarity of exposition, we assume this simple scoring scheme in setting up the definitions. We address the issue of extending the results to more complex scoring schemes in the next section.

We say that (x, y, z) is an *alignment vector for* $a_i \cdots a_k$ *and* $b_j \cdots b_l$, if there is an alignment path between the vertices $(i-1, j-1)$ and (k, l) in $G_{a,b}$ with x matches, y mismatches, and z indels. In Figure 2, $(3, 1, 4)$ is an alignment vector corresponding to the path shown in the figure. We denote by $AV_{i,j,k,l}(a, b)$ the set of all such alignment vectors, i.e.

$$AV_{i,j,k,l}(a, b) = \{(x, y, z) \mid (x, y, z) \text{ is an alignment vector for } a_i \cdots a_k \text{ and } b_j \cdots b_l\} .$$

Similarly we call (x, y, z) an *alignment vector* if it is an alignment vector for some pair $a_i \cdots a_k$ and $b_j \cdots b_l$. We define $AV(a, b)$ as the set of all alignment vectors, i.e.

$$AV(a, b) = \bigcup_{\substack{i \le k, \\ j \le l}} AV_{i,j,k,l}(a, b) \tag{1}$$

An alignment vector (x, y, z) has a score defined by δ, and μ:

$$SCORE(x, y, z) = x - \delta y - \mu z \tag{2}$$

The maximum score between segments $a_i \cdots a_k$ and $b_j \cdots b_k$ is the score of an alignment vector whose score is the maximum among all the alignment vectors between these two sequences:

$$S_{\delta,\mu}(a_i \cdots a_k, b_j \cdots b_k) = \max\{SCORE(x, y, z) \mid (x, y, z) \in AV_{i,j,k,l}(a, b)\} \tag{3}$$

In this paper, we denote by $\mathcal{P}^*$ the optimum value of problem $\mathcal{P}$. Local Alignment (LA) problem seeks for two segments with the highest similarity score LA^* as defined by

$$LA^*_{\delta,\mu}(a, b) = \max_{\substack{i \le k, \\ j \le l}} \{S_{\delta,\mu}(a_i \cdots a_k, b_j \cdots b_k)\} = \max_{\substack{i \le k, \\ j \le l}} \{SCORE(x, y, z) \mid (x, y, z) \in AV_{i,j,k,l}(a, b)\}$$

The same objective can also be expressed using the definition in (1) of the set of alignment vectors $AV(a, b)$ as

$$LA^*_{\delta,\mu}(a, b) = \max\{SCORE(x, y, z) \mid (x, y, z) \in AV(a, b)\} \tag{4}$$

In other words, the LA problem aims to find an alignment vector with highest score, or equivalently a directed path with largest weight in $G_{a,b}$.

Before introducing normalization of scores, we first define a length function with respect to some positive constant L as

$$LENGTH_L(a_i \cdots a_k, b_j \cdots b_l) = (k - i + 1) + (l - j + 1) + L .$$

A *normalized score* (with respect to L) NS_L of two segments $a_i \cdots a_k$, $b_j \cdots b_l$ is the ratio of their maximum score to the value of $LENGTH_L$ for these segments:

$$NS_{\delta,\mu,L}(a_i \cdots a_k, b_j \cdots b_l) = \frac{S_{\delta,\mu}(a_i \cdots a_k, b_j \cdots b_l)}{LENGTH_L(a_i \cdots a_k, b_j \cdots b_l)} \tag{5}$$

Normalized Local Alignment (NLA) problem seeks for two segments $a_i \cdots a_k$ and $b_j \cdots b_l$ for which the normalized score is the highest among all possible pairs of segments as expressed below:

$$NLA^*_{\delta,\mu,L}(a, b) = \max_{\substack{i \le k, \\ j \le l}} \{NS_{\delta,\mu,L}(a_i \cdots a_k, b_j \cdots b_l)\}$$

Observe that if (x, y, z) is an alignment vector for $a_i \ldots a_k$ and $b_j \ldots b_l$ then

$$(k - i + 1) + (l - j + 1) = 2x + 2y + z$$

Using this relation, we see that the function $LENGTH_L$ can be given on the set of alignment vectors $(x, y, z) \in AV(a, b)$ by the expression

$$LENGTH_L(x, y, z) = 2x + 2y + z + L \tag{6}$$

We can define the objective of the NLA problem in the domain of alignment vectors by using definitions in (1), (3), (5), and (6) as

$$NLA^*_{\delta,\mu,L}(a, b) = \max \left\{ \frac{SCORE(x, y, z)}{LENGTH_L(x, y, z)} \mid (x, y, z) \in AV(a, b) \right\} \tag{7}$$

Figure 3 shows some possible problem cases for LA for which NLA discriminates an alignment with higher percent matches from the one determined by the LA problem. Part (i) includes an example for the mosaic effect, and parts (ii), and (iii) have examples with non-overlapping and overlapping alignments respectively. In each case, the shorter alignment(s) with a score of 80 has a higher normalized score for $L < 600$ than the longer alignment, whose score is 120.

3 Algorithms

The alignment problems we define by stating their objectives in the previous section are clearly optimization problems of linear functions over the same domain. In other words, using equations (2) and (6), and definitions (4) and (7) we can rewrite LA and NLA as the following maximization problems :

$$\begin{array}{lcll} LA_{\delta,\mu}(a, b) & : & maximize\ x - \delta y - \mu z & \text{s.t.}(x, y, z) \in AV(a, b) \\ NLA_{\delta,\mu,L}(a, b) & : & maximize\ \frac{x - \delta y - \mu z}{2x + 2y + z + L} & \text{s.t.}(x, y, z) \in AV(a, b) \end{array}$$

For a given λ, we define a problem which we call *the parametric local alignment problem*

$$LA_{\delta,\mu,L}(\lambda)(a, b) \quad : \quad maximize\ x - \delta y - \mu z\ -\ \lambda(2x + 2y + z + L) \qquad \text{s.t. } (x, y, z) \in AV(a, b)$$

Since the formal parameters in the problem descriptions are the same, in the rest of the paper we will use LA, NLA and $LA(\lambda)$ instead of $LA_{\delta,\mu,L}(a, b)$, $NLA_{\delta,\mu,L}(a, b)$, and $LA_{\delta,\mu,L}(\lambda)(a, b)$, respectively.

As we propose next, a parametric local alignment problem can be described in terms of local alignment problem.

Proposition 1 *For $\lambda < \frac{1}{2}$, the optimum value $LA^*(\lambda)$ of the parametric LA problem can be formulated in terms of the optimum value LA^* of an LA problem.*

Proof

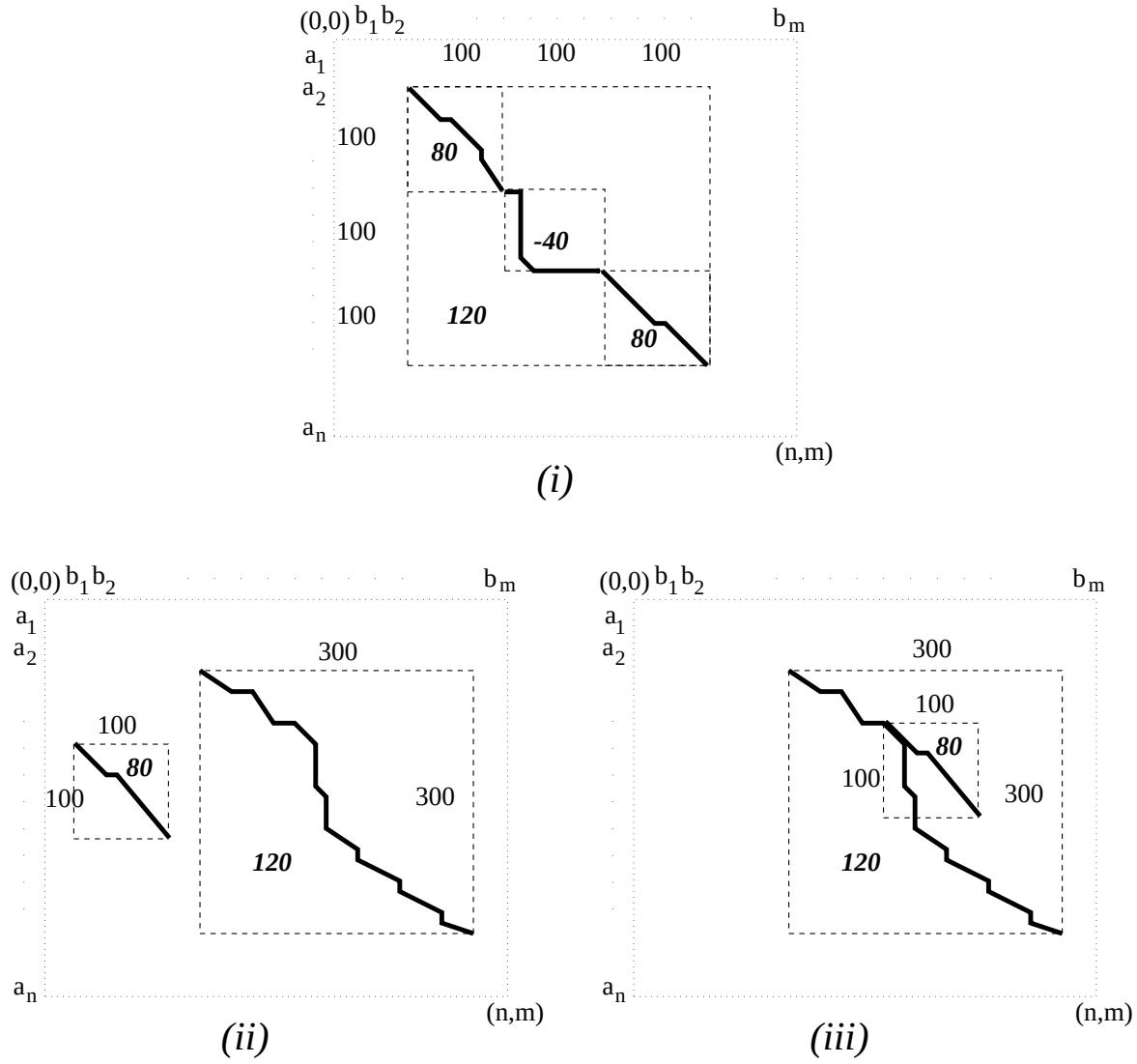

Figure 3: Mosaic and shadow effects. (i) mosaic effect, (ii) shadow effect (non-overlapping alignments), (iii) shadow effect (overlapping alignments). The numbers written in italic are the scores of alignments identified by the corresponding rectangles. The other numbers are the side lengths of the rectangles. The normalized score of the shorter alignment(s) is $\frac{80}{200+L}$ while that of the longer alignment is $\frac{120}{600+L}$.

The objective of the parametric problem is

$$\begin{aligned} LA^*(\lambda) &= \max\{(1-2\lambda)x - (\delta+2\lambda)y - (\mu+\lambda)z - \lambda L\} \\ &= (1-2\lambda)\max\left\{x - \frac{\delta+2\lambda}{1-2\lambda}y - \frac{\mu+\lambda}{1-2\lambda}z\right\} - \lambda L \\ &= (1-2\lambda)LA^*_{\delta',\mu'}(a,b) - \lambda L \\ &\quad \text{where } \delta' = \frac{\delta+2\lambda}{1-2\lambda}, \quad \mu' = \frac{\mu+\lambda}{1-2\lambda}. \end{aligned} \tag{8}$$

Thus, computing $LA^*(\lambda)$ involves solving the local alignment problem $LA_{\delta',\mu'}(a,b)$, and performing some simple arithmetic afterwards. □

Note that since δ, μ and L are positive, for any alignment vector (x', y', z'), its normalized score

$$\lambda = \frac{x' - \delta y' - \mu z'}{2x' + 2y' + z' + L} < \tfrac{1}{2}$$

Dinkelbach's algorithm (Dinkelbach, 1967) can be used to solve *NLA* . Dinkelbach has developed a general algorithm which uses the *parametric method* of an optimization technique known as *fractional programming*. The algorithm is applicable to optimization problems which involve a ratio of two functions over the same domain where the function in the denominator is assumed to be positive. The thesis of the parametric method applied to the case of alignment maximization problems implies that the optimal solution to *NLA* can be achieved via a series of optimal solutions of $LA(\lambda)$ for different λ . The central result is that

$$\lambda = NLA^* \quad \text{iff} \quad LA^*(\lambda) = 0 \ .$$

That is, an alignment vector a has the optimum normalized score λ iff a is an optimal alignment vector for the parametric problem $LA(\lambda)$ whose optimum value is zero. A proof of this essential property of the parametric method is given by Sniedovich (1992). Craven (1988) and Sniedovich (1992) explain various other interesting properties of Dinkelbach's algorithm and fractional programming.

Dinkelbach algorithm for *NLA* problem is shown in Figure 4. The algorithm starts with an initial value for λ and repeatedly solves $LA(\lambda)$. At each instance of the parametric problem, an optimal alignment vector (x, y, z) of $LA(\lambda)$ yields a ratio (normalized score) for *NLA*. This new ratio is either equal to λ, in which case it is optimum, or larger than λ . If it is equal to λ then the algorithm terminates. Note that in this case $LA^*(\lambda) = 0$ since the optimal alignment vector of the last iteration has the normalized score λ . Otherwise, the ratio is taken to be the new value of λ and $LA(\lambda)$ is solved again. When continued in this fashion, convergence to NLA^* is guaranteed. Another way to explain the behavior of the algorithm is as follows. It iteratively modifies the scores in such a way that the optimal non-normalized local alignment under the set of converged scores is also the optimal normalized alignment under the original scores.

The parametric problem in this algorithm can be solved using the Smith-Waterman algorithm. An optimal alignment vector needs to be computed along with optimal score for the parametric problem of the Dinkelbach algorithm. Position of an optimal alignment may also be desired. These can be done by extending the Smith-Waterman algorithm to include, at each entry of the score matrix, information about the alignment vector corresponding to an optimal alignment path which ends at that node, and the starting node-position of the path. This additional information can be

```
Algorithm Dinkelbach

Pick an arbitrary alignment vector (x, y, z) ∈ AV(a, b) ,

            λ* ← (x − δy − μz) / (2x + 2y + z + L)

Repeat

   λ ← λ*

   Using Prop.1, solve LA(λ) and obtain an optimal alignment vector (x, y, z)

   λ* ← (x − δy − μz) / (2x + 2y + z + L)

Until λ* = λ

Return(λ*)
```

Figure 4: `Dinkelbach` algorithm for NLA.

carried over and updated along with the optimal score updates without an increase in the asymptotic space and time complexity. The resulting space complexity of solving NLA by this algorithm is $O(m)$. The resulting time complexity is the product of the number of iterations and, the time complexity of the Smith-Waterman algorithm. Although experimental results suggest that the number of iterations is small on average, no satisfactory theoretical average-case/worst-case bound for the growth of the number of iterations has been established.

We show next that a provably better time complexity result can be achieved by using Megiddo's technique.

Megiddo (1979) introduced a general technique on how to use a given algorithm for optimizing a linear function in order to develop an algorithm for an optimization problem which involves a ratio of two linear functions over the same domain. If we apply his technique to NLA computation, then the resulting algorithm is an LA algorithm which assumes a score of $1-\lambda$ for a match, penalties of $\delta-\lambda$, and $\mu-\lambda$ for a mismatch and an indel, respectively. λ is treated as a variable, not a constant. That is, the algorithm is the same LA algorithm except that the coefficients are not simple constants but linear functions of the parameter λ. Instead of repeatedly solving $LA(\lambda)$ with increasing values of λ as in the Dinkelbach algorithm, this alternative solution simulates the given LA algorithm over the coefficients. Additions of these linear functions are linear and can be computed immediately, but comparisons among them need to be done with some care. The algorithm keeps track of the interval in which the optimum value NLA^* lies. This is essential because comparisons in the given LA algorithm now correspond to those among linear functions, and outcomes may vary depending on interval under consideration for λ.

The algorithm starts with the initial interval $[-\infty, +\infty]$ for NLA^*. If the functions to be compared intersect, then their intersection point λ', *"a critical value"* of λ, determines two subintervals of the initial interval. In calculating which of the two subintervals contains NLA^*, the LA algorithm

is called for help, and problem $LA(\lambda')$ is solved. The new interval and the result of the comparison are determined from the sign of the optimum value $LA^*(\lambda')$ as will be explained later. The algorithm returns a linear function of λ and a final interval by which the local maximum of the function can be computed. With this technique, if LA is solvable using $O(p(n))$ comparisons and $O(q(n))$ additions then NLA can be solved in time $O(p(n)(p(n)+q(n)))$. If we choose the Smith-Waterman algorithm to simulate then the time complexity of the resulting algorithm is $O(n^2m^2)$.

Megiddo (1979) also showed that for some problems the critical values of λ can be precomputed. In such cases these values give us the possible candidates for the end-points of the smallest interval which eventually contains the optimum value (ratio). (In some applications, even all candidate optimum values can be precomputed efficiently (Arslan and Eğecioğlu, 1999; Arslan and Eğecioğlu, 2000).

Whenever this can be done, *binary search* can be used to find the optimum value. For the alignment problems in this paper: If $LA^*(\lambda) = 0$, then $\lambda = NLA^*$, and an optimal alignment vector of $LA(\lambda)$ is also an optimal solution of NLA . On the other hand, if $LA^*(\lambda) > 0$, then a larger λ, and if $LA^*(\lambda) < 0$, then a smaller λ should be tested (i.e. problem $LA(\lambda)$ should be solved with a different value of λ). This procedure continues until the "correct" value NLA^* is found. Let λ' be the largest value in the set for which $LA^*(\lambda')$ is less than or equal to zero. Then an optimal alignment vector of $LA(\lambda')$ yields the optimum value NLA^* . This way, number of invocations of LA algorithm is much smaller than that of the solution which uses the simulation idea. This technique was used in problems such as minimum ratio cycles, and minimum ratio spanning trees (Megiddo, 1979), and normalized edit distance (Arslan and Eğecioğlu, 1999; Arslan and Eğecioğlu, 2000).

It does not seem feasible to precompute critical or candidate values for the optimum value of NLA . However, we will show that an efficient search for the optimum value is still possible by using the fact that any two distinct candidate values for NLA^* are not arbitrarily close to each other if the scores are rational. A similar observation was used for the computation of normalized edit distance by Arslan and Eğecioğlu (2000).

Let $Q(a,b)$ be the set of possible values for NLA^* . That is,

$$Q(a,b) = \left\{ \frac{x - \delta y - \mu z}{2x + 2y + z + L} \mid (x,y,z) \in AV(a,b) \right\}$$

Proposition 2 *Let*

$$\sigma = \min\{ \ |q_1 - q_2| \ \mid q_1, q_2 \in Q(a,b), q_1 \neq q_2 \}$$

denote the smallest gap in $Q(a,b)$ and assume $\delta = \frac{p}{q}$ and $\mu = \frac{r}{s}$ are rational. Then

$$\sigma \geq \frac{1}{qs(m+n+L)^2}.$$

Proof Suppose $q_1, q_2 \in Q(a,b)$ be two normalized scores of the alignment vectors (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively, where $q_1 > q_2$. Then

$$\sigma \geq \frac{x_1 - \delta y_1 - \mu z_1}{2x_1 + 2y_1 + z_1 + L} - \frac{x_2 - \delta y_2 - \mu z_2}{2x_2 + 2y_2 + z_2 + L}$$

Observe that for two positive rationals $\frac{p_1}{q_1} > \frac{p_2}{q_2} \Rightarrow \frac{p_1}{q_1} - \frac{p_2}{q_2} \geq \frac{1}{q_1 q_2}$. Also, for any alignment vector $(x, y, z) \in AV(a, b)$, since $2x + 2y + z \leq m + n$ we have

$$\begin{aligned} \sigma &\geq \frac{1}{qs}\left[\frac{qsx_1 - psy_1 - qrz_1}{2x_1 + 2y_1 + z_1 + L} - \frac{qsx_2 - psy_2 - qrz_2}{2x_2 + 2y_2 + z_2 + L}\right] \\ &\geq \frac{1}{qs(m + n + L)^2} \end{aligned} \tag{9}$$

□

We propose the following algorithm `RationalNLA` for the *NLA* problem with rational penalties (Figure 5). The algorithm first computes the smallest possible gap σ between any two distinct values for *NLA* (Proposition 2). It maintains an interval, $[e, f]$, such that the optimum value of *NLA* lies in $[e\sigma, f\sigma]$ where e, and f are appropriate integer values. Initially e is set to zero, and f is set to $\frac{1}{2}\sigma^{-1}$ since NLA^* is in $[0, \frac{1}{2})$. `RationalNLA` iteratively solves a parametric local alignment problem with parameter $k\sigma$ where k is the median of integers in $[e, f]$. At each iteration the interval is updated according to the sign of the optimum value of the parametric problem as explained in Megiddo's technique. The effective search space is the integers in $[e, f]$ and each iteration reduces this space by half. The iterations end whenever the optimum value for the parametric local alignment problem is zero upon which the algorithm terminates by returning the parameter $k\sigma$ as the optimum value of *NLA*, or whenever there remains no integers between e and f. In the latter case, the algorithm solves a parametric local alignment problem with parameter $f\sigma$. An optimal solution of this parametric problem yields the optimum normalized local alignment score (the optimum value of *NLA*).

The invariant for the while loop is that $e \leq f$, and NLA^* is in $[e\sigma, f\sigma]$. We can prove that it holds by induction on the number of iterations. At the beginning (iteration zero) the invariant is true since e and f are initialized to zero and $\frac{1}{2}\sigma^{-1}$, respectively, and NLA^* is in $[0, \frac{1}{2})$. The proof of the inductive step follows from the discussions of Megiddo's search technique. The algorithm returns the parameter value if in one of the iterations the optimum value of the local parametric alignment problem is zero in which case the algorithm is correct. Otherwise, the while-loop terminates with the following conditions being true: $e \leq f$ and $e + 1 \geq f$ (i.e. $e = f$ or $e + 1 = f$), and NLA^* is in $[e\sigma, f\sigma]$. Since the minimum distance between any two possible distinct values for NLA^* is at least σ,

(i) either $NLA^* = f\sigma$,

(ii) or NLA^* is in $[e\sigma, f\sigma)$ in which case there is only one possible value for NLA^* in $[e\sigma, f\sigma)$.

In both cases, an optimal alignment vector (x, y, z) for the parametric local alignment problem with parameter $f\sigma$ yields the optimum value NLA^* because of the fact that each new solution to a parametric problem yields a ratio no worse than the parameter value as pointed out in the description of the Dinkelbach algorithm.

Theorem 1 *If algorithm A computes LA^* and obtains an optimal alignment vector with time complexity $T(n, m)$, then* NLA^* *can be computed in time $O(T(n, m) \log n)$ and using (asymptotically) the same space required by algorithm A provided that δ and μ are rational.*

Proof The while loop in `RationalNLA` iterates $O(\log(\frac{1}{2}qs(m + n + L)^2))$ times because the space on which binary search is performed is included in the set of integers in the range $[0, \frac{1}{2}qs(m+n+L)^2]$.

Algorithm RationalNLA

$\sigma \leftarrow \frac{1}{qs(m+n+L)^2}$ where $\delta = \frac{p}{q}$, and $\mu = \frac{r}{s}$ (Prop. 2)

$[e, f] \leftarrow [0, \frac{1}{2}qs(m+n+L)^2]$

While $(e+1 < f)$ do

  $k \leftarrow \lfloor (e+f)/2 \rfloor$

  Using Prop.1, solve $LA(k\sigma)$ and obtain an optimal alignment vector (x, y, z)

  $v \leftarrow \frac{x-\delta y-\mu z}{2x+2y+z+L}$

  if $v = 0$ then return$(k\sigma)$

  else if $v < 0$ then $f \leftarrow k$

  else $e \leftarrow k$

End {while}

Using Prop.1, solve $LA(f\sigma)$ and obtain an optimal alignment vector (x, y, z)

Return$\left(\frac{x-\delta y-\mu z}{2x+2y+z+L}\right)$

Figure 5: *NLA* algorithm `RationalNLA` for rational scores.

Solving each parametric problem takes $T(n,m)$ time using algorithm A since it involves only a local alignment computation and some simple arithmetic. The remaining steps take constant time. Therefore the resulting time complexity is $O(T(n,m)\log(\frac{1}{2}qs(m+n+L)^2))$. The space complexity is the same as that of algorithm A . □

The Smith-Waterman algorithm can be used to find the local alignment vectors and hence to solve the parametric local alignment problems invoked by `RationalNLA`.

Corollary 1 *Normalized local alignment of sequences of length n and m can be computed in $O(nm\log n)$ time and $O(m)$ space.*

The ideas in the Dinkelbach algorithm or algorithm `RationalNLA` are not restricted to a particular scoring scheme. Under any given scoring scheme, provided that the parametric LA problems in these algorithms can be formulated in terms of an LA problem, these algorithms can be modified so that they present a solution to NLA problem. Furthermore, if scores/penalties are rational, and solving a parametric problem and obtaining an optimal solution (alignment vector) take asymptotically the same time as that of the underlying LA algorithm, then the complexity results for `RationalNLA` of Theorem 1 hold. We address two particularly important cases of scoring schemes : *affine gap penalties*, and *arbitrary score matrices.*

Sometimes insertion or deletion of a block of symbols called a *gap* is treated differently than a stream of single-symbol indels. Affine gap penalty for a gap of length k is

$$\alpha + \mu k$$

where α is a *gap open penalty* and μ is an indel penalty. In this case, we may use a 4-tuple (x,y,z,g) to represent an alignment vector with which the new component g is the number of gaps. For example, $(3,1,4,2)$ is the alignment vector for the alignment path shown in Figure 2. The alignment vector has two gaps one of which is a single delete, and the other is a block of three inserts. The definition of the length function $LENGTH_L$ does not change under this scoring scheme. The score of an alignment vector can be rewritten as

$$SCORE(x,y,z,g) = x - \delta y - \mu z - \alpha g \ .$$

In some applications, score of a given operation varies depending on the individual symbols involved in the operation (e.g., protein sequence comparison). In this case, we may decide to define the alignment vector such that it includes as a component frequency of each operation. Let $i-$, $-i$ denote respectively the deletion and insertion of the ith symbol, and ij denote the substitution of the jth symbol for the ith symbol of the alphabet Σ . For a given operation e, let s_e represent the score, and f_e represent the frequency of this operation. If $u = |\Sigma|$ then for a given alignment vector a where

$$a =< f_{1-}, f_{2-}, \ldots, f_{u-}, f_{-1}, f_{-2}, \ldots, f_{-u}, f_{11}, f_{12}, \ldots, f_{1u}, \ldots, f_{u1}, f_{u2}, \ldots, f_{uu} > \ ,$$

the score and length functions can be defined as

$$\begin{aligned} SCORE(a) &= \sum_{ij} s_{ij} f_{ij} + \sum_i s_{i-} f_{i-} + \sum_i s_{-i} f_{-i} \\ LENGTH_L(a) &= 2\sum_{ij} f_{ij} + \sum_i f_{i-} + \sum_i f_{-i} + L \end{aligned}$$

One can verify that in both of these cases, a parametric LA problem can easily be formulated in terms of an LA problem under that particular scoring scheme, and our results hold.

4 Implementation and Test Results

We have chosen to implement the Dinkelbach algorithm for *NLA* computation (affine gap penalties) since this algorithm has a good performance in practice. We have modified the Smith-Waterman algorithm (for affine gaps) to obtain and carry along the alignment information through the nodes. In our implementation we have used $LENGTH_L$ value of the alignment vectors as a tie breaker. We select an alignment with the largest $LENGTH_L$ value in case there are more than one optimal alignments ending in the same node. That is, we favor the alignment with the largest $LENGTH_L$ value among the alignments with the same normalized score since for two alignments with the same normalized score, the one with larger $LENGTH_L$ value has the higher (non-normalized) score which may be preferred over others (The program can be obtained by contacting Arslan, A.N.). In our tests, the algorithm never required more than 9 invocations of the Smith-Waterman algorithm, and in the majority of cases it took $3-5$ invocations to solve a single *NLA* problem.

Once optimal segments are found for one *NLA* problem, one may want to continue with more *NLA* computations after masking these segments in the two sequences. For this purpose, we have developed algorithm `RepeatedDinkelbach`. With each alignment between $a_i \dots a_k$ and $b_j \dots b_l$, we store a pair whose first component is the alignment vector (x, y, z, g) and second component is the alignment position (i, j, k, l) . We have used a queue Q to store alignments generated by the iterations of the Dinkelbach *NLA* algorithm so that a new *NLA* computation picks as the initial alignment the last alignment in Q which does not overlap with the alignment reported in the last iteration. This way we improve the average number of iterations per *NLA* computation. `RepeatedDinkelbach` continues generating alignments until no alignment whose normalized score is larger than a given threshold score T can be found in unmasked regions of the sequences. This termination condition is easy to implement since the normalized scores are decreasing as they are reported. Another alternative would be to let the algorithm run until there remains no more alignments with positive score. We have also implemented a version of the algorithm which first masks a set of regions as a pre-processing step. This allows us to explicitly stop the *NLA* computations at any time we want, and resume the computation of alignments from where it (almost) left using the second algorithm.

We have tested our algorithms with various values of L . We observe that if L is large we obtain alignments with high scores but low normalized scores, while if L is small then the resulting alignments have high normalized scores but they may be short and less interesting biologically. In other words, as the value of L increases our algorithm finds longer optimal alignments for a particular instance of the problem. It is difficult to determine a value for L which performs well in (almost) every case because a proper value is data-dependent. If the highest normalized score (with respect to the current value of L) belongs to an alignment that is too short to be biologically interesting then we need to increase the value of L to favor the longer (biologically interesting) alignments. For example for the alignments in Figure 3, L has to be at least 600 so that the longer alignment wins over the shorter one. If alignments returned as optimal do not have sufficiently high normalized scores then a smaller values of L should be tried. One needs to experiment various values for L for a particular instance of sequence alignment. Another way to get rid of unwanted short alignments can be to mask the corresponding regions and rerun the algorithm. If we decide to do so we need to be sure that these regions do not take part in desired alignments. As a common practice in sequence alignment, we first masked the repeats by RepeatMasker (http://ftp.genome.washington.edu/RM/RepeatMasker.html) before running our algorithm. These biologically uninteresting regions may have high normalized scores. They may

become part of unwanted short alignments. Therefore hiding repeats may help eliminate short alignments to be output as optimal by our algorithm. To visualize the difference between different approaches to sequence alignment, we represented every area of similarity as a rectangle rather than as a diagonal in conventional drawings of dot-matrices. Rectangles in the figures show the segments involved in the alignments. In Figures 6 and 7 the alignment regions returned by Smith-Waterman algorithm are shown using dotted lines whereas those determined by post-processing algorithm by Zhang *et al.* (1999) are distinguished by dashed lines. Rectangles with thick lines are the ones obtained by our algorithm. We have included percent matches (number of matches divided by the average length of the segments) for the alignments we have found. Our algorithm captures the regions found by these algorithms but provides more "granularity" in representing the most similar fragments of the aligned regions. To achieve even higher level of granularity one can either reduce the threshold T for reported alignments or vary L at different iterations of the algorithm. As expected, the regions not included in found normalized local alignments show little similarity: the degree of similarity "outside" the boxes in Figures 6 and 7 is usually below 35%.

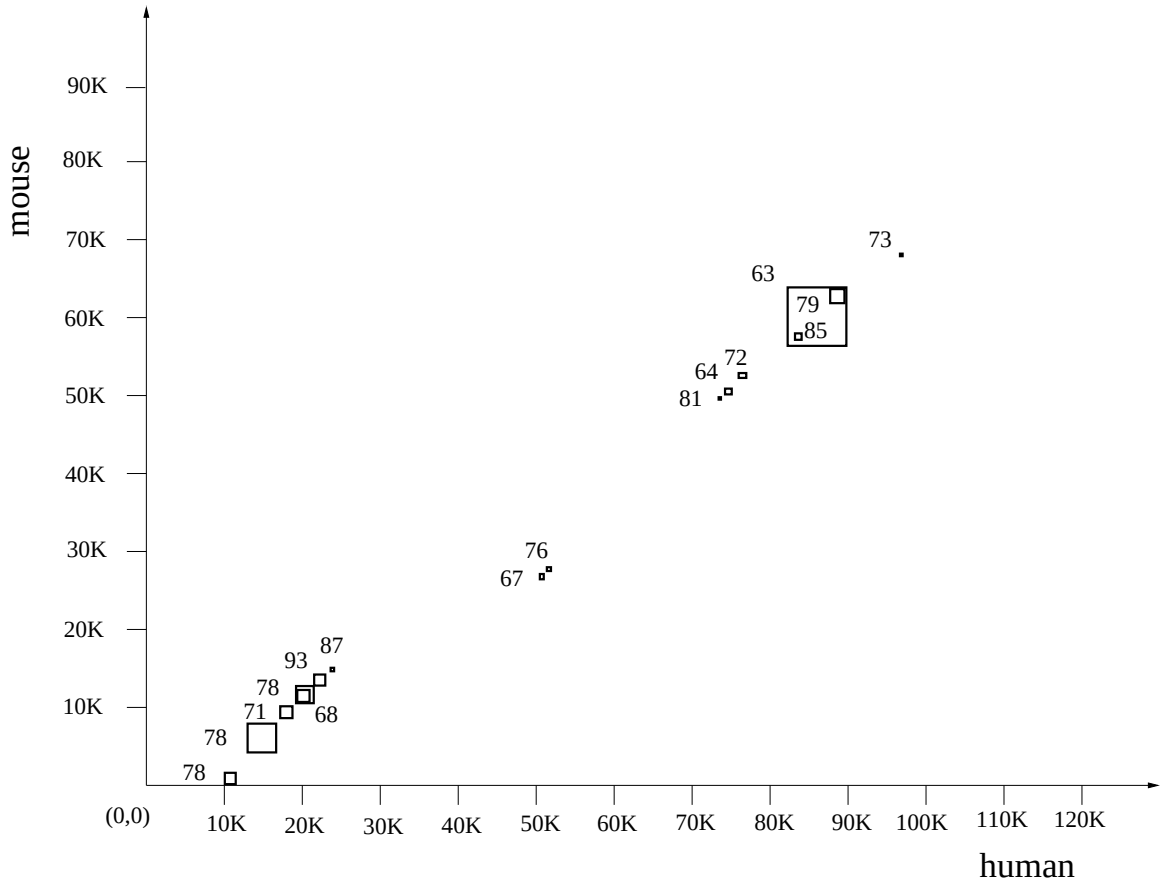

Figure 6: Normalized local alignments of orthologous human (GenBank Acc. No. AF030876) and mouse (GenBank Acc. No. AF121351) genomic sequences ($L = 2000$, $\delta = 1$, $\alpha = 6$, and $\mu = 0.2$).

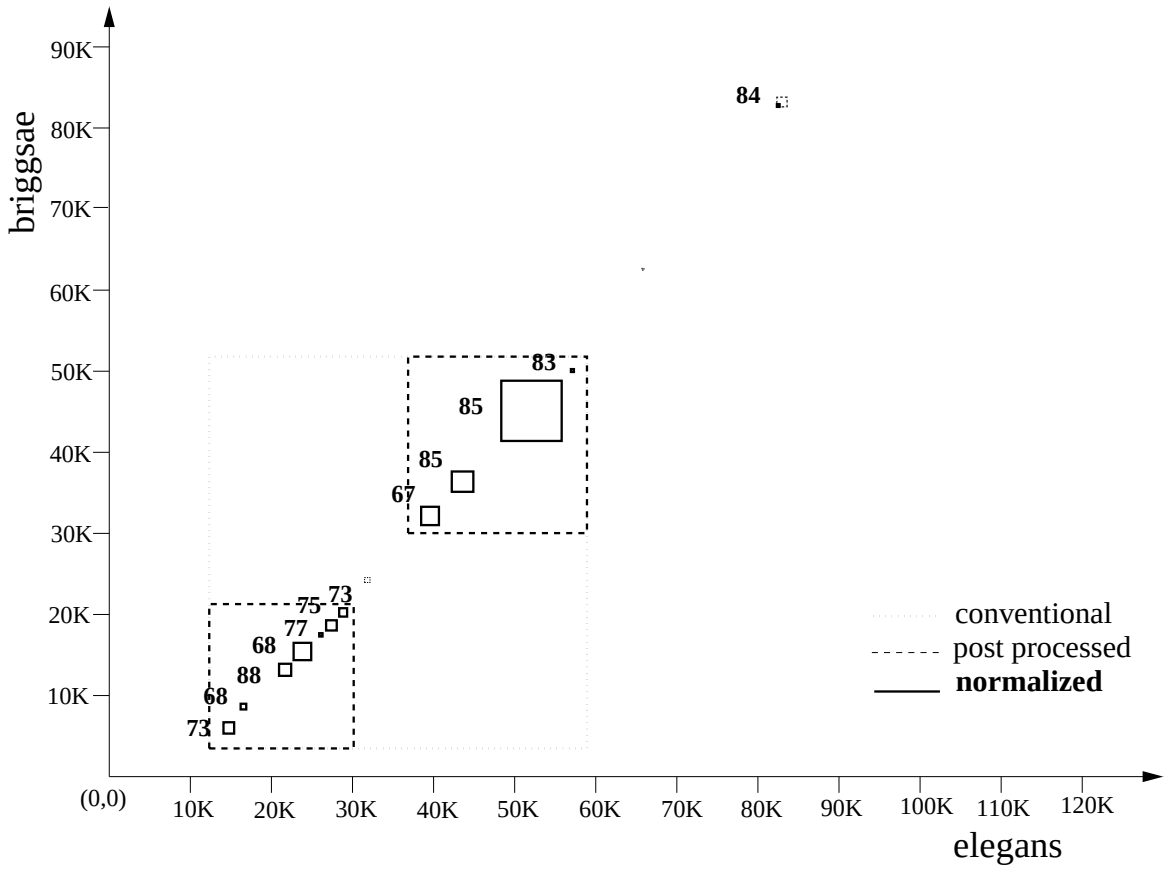

Figure 7: Comparison of normalized local alignments of *bli-4* locus in *C. elegans* and *C.briggsae* with conventional local alignments and post-processed local alignments as described by Zhang *et al.* (1999) ($L = 2000$, $\delta = 1$, $\alpha = 6$, and $\mu = 0.2$).

5 Conclusions

The arrival of long genomic sequences raises new challenges in sequence comparison. In particular, the traditional tools for computing and representing alignments may not be suitable for genomic-scale sequence comparison. These challenges were recently addressed by Schwartz *et al.* (2000) who introduced the *Percent Identity Plots* or *PIPs*. PIPs are compact and convenient substitutes for dot-matrices that, in addition to revealing similar segments, reflect the percent of similarity between different segments of compared sequences. Our normalized local approach is conceptually similar to this approach in an attempt to find the regions with the highest percent of similarity.

The undesirable properties of linear scoring in sequence alignment were first revealed by Altschul and Erickson (1986) who proposed different non-linear scoring functions. They also noticed that alignments with non-linear scoring functions are difficult to compute in practice. The deficiency of linear scoring functions are well-known in other application domains of dynamic programming. In particular, non-linear scoring functions lead to better practical algorithms for Speech Recognition and Recognition of Hand-Written Texts (Vidal *et al.*, 1995). In computational molecular biology, Pearson (1995) and Shpaer *et al.* (1996) tried to remedy the deficiencies of the linear scoring functions by re-normalization of the Smith-Waterman scores at the post-processing stage. This re-normalization led to significant improvement in the selectivity of the database searches. Although these approaches are similar in spirit to our work, we emphasize the important difference: re-normalizations rearrange the ranked list of the Smith-Waterman scores but do not affect the Smith-Waterman algorithm itself. It is possible that an alignment found by normalized local alignment algorithm is overlapping with no alignments given by Smith-Waterman algorithm.

Pearson, 1995 (Pearson, 1995), Shpaer et al., 1996 (Shpaer *et al.*, 1996) and Brenner et al.,1998 (Brenner *et al.*, 1998) made the comparative analysis of FASTA, BLAST and the Smith-Waterman algorithm for functional protein classification. Abdueva et al. 2001 (Abdueva *et al.*, 2001) used their test framework to study the effect of alignment length on sensitivity of database search. The preliminary results of this work demonstrate that normalization improves the functional protein classification.

Some sequence comparison practitioners have been using a few runs of the Smith-Waterman algorithm with varied gap penalties to arrive to a biologically adequate alignment. However, the choice of gap penalties in such searches remained largely heuristic. Our algorithm for normalized sequence alignment mimics this approach but provides a rigorous justification for choosing parameters in different runs of the Smith-Waterman algorithm.

Although the normalized local alignment approach proved to be successful in our preliminary tests, a number of questions remain unsolved. Most importantly, the statistics of normalized local alignment is poorly understood. The statistical questions associated with the classical local alignment are so complex (Arratia *et al.*, 1990; Waterman and Gordon, 1990; Waterman and Vingron, 1994) that we did not even dare to try estimating statistical significance of normalized local alignment. Normalization helps eliminate the mosaic and shadow effects. The success depends on the value of L. It seems that no single choice of L eliminates all these unwanted effects, and reveals all the most important alignments at the same time. However, we can argue that there exists a value of L with which an important alignment may be detected by normalized alignment algorithm if it has sufficiently high normalized score. An important problem is that given an instance of sequence alignment, the rules governing the optimal choice of the parameter L are not yet well understood.

6 Acknowledgements

We are grateful to Diana Abdueva, Nickolai Alexandrov, Steven Altschul, Mikhail Gelfand, Ben Koop, Martin Vingron, and Michael Waterman for helpful discussions, and to two anonymous referees for their constructive comments.

References

Abdueva, D., Solovyev, V. V., Trukhan, M., Pevzner, P. A. and Alexandrov, N. N. (2001). Normalized local alignment improves the functional characterization of proteins, *(in preparation)*.

Alexandrov, N. N. and Solovyev, V. V. (1998). Statistical significance of ungapped alignments, *Pacific Symposium on Biocomputing (PSB-98), (eds. R. Altman, A. Dunker, L. Hunter, T. Klein)* pp. 463–472.

Altschul, S. F. and Erickson, B. W. (1986). Locally optimal subalignments using nonlinear similarity functions, *Bulletin of Mathematical Biology*, **48**, 633–660.

Altschul, S. F. and Erickson, B. W. (1988). Significance levels for biological sequence comparison using nonlinear similarity functions, *Bulletin of Mathematical Biology*, **50**, 77–92.

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., W, M. and Lipman, D. J. (1997). Gapped Blast and Psi-Blast: a new generation of protein database search programs, *Nucleic Acids Research*, **25**, 3389–3402.

Arratia, R., Gordon, L. and Waterman, M. S. (1990). The Erdös-Renyi Law in distribution, for coin tossing and sequence matching, *The Annals of Statistics*, **18**, 539–570.

Arslan, A. N. and Eğecioğlu, Ö. (1999). An efficient uniform-cost normalized edit distance algorithm, *6th Symposium on String Processing and Information Retrieval (SPIRE'99), IEEE Comp. Soc.* pp. 8–15.

Arslan, A. N. and Eğecioğlu, Ö. (2000). Efficient algorithms for normalized edit distance, *Journal of Discrete Algorithms, Special Issue on Matching Patterns, Hermes Science Publications, (in press)* .

Bafna, V. and Huson, D. (2000). The conserved exon method for gene finding, *Proceedings of the Eight International Conference on Intelligent Systems for Molecular Biology, La Jolla, California* .

Batzoglou, S., Pachter, L., Mesirov, J., Berger, B. and Lander, E. (2000). Comparative analysis of mouse and human dna and applications to exon prediction, *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology (RECOMB-99)), Tokyo, Japan.*

Birney, E., Thompson, J. D. and Gibson, T. J. (1996). PairWise and SearchWise: finding the optimal alignment in a simultaneous comparison of a protein profile against all DNA translation frames, *Nucleic Acids Res.*, **24**, 1730–1739.

Brenner, S. E., Chotia, C. and Hubbard, T. J. (1998). Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships, *Proc. Natl. Acad. Sci. USA*, **95**, 6073-6078.

Craven, B. D. (1988). *Fractional Programming*, Helderman Verlag, Berlin.

Dinkelbach, W. (1967). On nonlinear fractional programming, *Management Science*, **13**, 492–498.

Eğecioğlu, Ö. and Ibel, M. (1996). Parallel algorithms for fast computation of normalized edit distances, *Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing (SPDP'96)* pp. 496–503.

Gelfand, M. S., Mironov, A. A. and Pevzner, P. A. (1996). Gene recognition via spliced sequence alignment, *Proceedings of the National Academy of Sciences*, **93**, 9061–9066.

Goad, W. B. and Kanehisa, M. I. (1982). Pattern recognition in nucleic acid sequences. i. a general method for finding local homologies and symmetries, *Nucleic Acid Research*, **10**, 247–263.

Huang, X., Pevzner, P. A. and Miller, W. (1994). Parametric recomputing in alignment graph, *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, Asilomar, California* pp. 87–101.

Marzal, A. and Vidal, E. (1993). Computation of normalized edit distances and applications, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(9), 926–932.

Megiddo, N. (1979). Combinatorial optimization with rational objective functions, *Mathematics of Operations Research*, **4**, 414–424.

Novichkov, P. S., Gelfand, M. S. and Mironov, A. A. (2000). Prediction of the exon-intron structure by comparison sequences, *Molecular Biology*, **34**, 200–206.

Oommen, B. J. and Zhang, K. (1996). The normalized string editing problem revisited, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(6), 669–672.

Pachter, L., Batzoglou, S., Spitkovsky, V. I., Lander, E. S., Berger, B. and Kleitman, D. J. (1999). A dictionary based approach for gene annotation, *Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB-99)), Lyon, France* pp. 285–294.

Pearson, W. R. (1995). Comparison of methods for searching protein sequence databases, *Protein Science*, **4**, 1145–1160.

Schwartz, S., Zhang, Z., Fraser, K. A., Smit, A., Riemer, C., Bouck, J., Gibson, R., Hardisson, R. and Miller, W. (2000). Pipmaker - a web server for aligning two genomic dna sequences, *Genome Research*, **10**, 577–586.

Sellers, P. H. (1984). Pattern recognition in genetic sequences by mismatch density, *Bulletin of Mathematical Biology*, **46**, 501–504.

Shpaer, E. G., Robinson, M., Yee, D., Candlin, J. D., Mines, R. and Hunkapiller, T. (1996). Sensitivity and selectivity in protein similarity searches: a comparison of smith-waterman in hardware to blast and fasta, *Genomics*, **38**, 179–191.

Smith, T. F. and Waterman, M. S. (1981). The identification of common molecular subsequences, *J. Mol. Biol.*, **147**, 195–197.

Sniedovich, M. (1992). *Dynamic Programming*, Marcel Dekker, New York.

Vidal, E., Marzal, A. and Aibar, P. (1995). Fast computation of normalized edit distances, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**, 899–902.

Waterman, M. S. (1995). *Introduction to Computational Biology*, Chapman and Hall, London.

Waterman, M. S. and Gordon, L. (1990). Multiple hypothesis testing for sequence comparison, *Computers in DNA. (Eds. G. Bell and T. Marr), Addison-Wesley* pp. 127–135.

Waterman, M. S. and Vingron, M. (1994). Rapid and accurate estimates of statistical significance for sequence database searches, *Proc. Natl. Acad. Sci. USA*, **91**, 4625–4628.

Zhang, Z., Berman, P. and Miller, W. (1998). Alignments without low-scoring regions, *J. Comput. Biol.*, **5**, 197–200.

Zhang, Z., Berman, P., Wiehe, T. and Miller, W. (1999). Post-processing long pairwise alignments, *Bioinformatics*, **15**, 1012–1019.